The Thesis Committee for Aastha Tripathi
Certifies that this is the approved version of the following thesis:

Analysis of Storage bottlenecks in Deep Learning models

APPROVED BY

SUPERVISING COMMITTEE:

Vijay Chidambaram Velayudhan Pillai, Supervisor

Christopher Rossbach

Analysis of Storage bottlenecks in Deep Learning models

by

Aastha Tripathi

Thesis

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Computer Science

The University of Texas at Austin

May 2019

Dedicated to my parents, Mr. Laxmikant Tripathi and Mrs. Manjula Tripathi, and my brother, Abhas Tripathi.

Acknowledgments

I would first of all want to thank my supervisor Prof. Vijay Chidambaram, Assistant Professor in the Department of Computer science at The University of Texas at Austin. He gave me the right combination of freedom to explore and independently work and at the same time guided me through whenever I was stuck. He guided me all through and it was great learning experience working with him. I am very grateful to be advised by him.

I would like to thank Prof. Christopher Rossbach, Professor in the Department of Computer science at The University of Texas at Austin for serving as the second reader for my thesis, giving me his valuable time and feedback. I would also really like to thank Jayashree Mohan, a Ph.D. student at The University of Texas at Austin who worked with me and provided me with her valuable time and guidance at different instances of the time. I am thankful to the other members of the Storage and Systems Lab for their valuable feedback.

I would also like to thank my friend Madhumitha Sakthi, a Ph.D. student at The University of Texas at Austin for providing her comments, for the discussions on the topic and for the enormous moral support throughout my thesis work.

I am grateful to my parents and family for believing in me and in what I pursue. I would like to thank my father Dr. Laxmikant Tripathi who has been ideal and motivated me to pursue a Masters degree and to explore research as thesis work. I would also like to thank a multitude of friends here at The University of Texas at Austin, who gave me moral support all throughout my Master's course.

Finally, I would like to thank God for giving me an opportunity to pursue Masters at The University of Texas at Austin, to work with Prof. Vijay Chidambaram, to work on the exciting work in this thesis, and for blessing me with a wonderful life.

Abstract

Analysis of Storage bottlenecks in Deep Learning models

Aastha Tripathi, M.S.C.S.
The University of Texas at Austin, 2019

Supervisor: Vijay Chidambaram Velayudhan Pillai

Deep Learning (DL) is gaining prominence and is widely used for a plethora of problems. DL models, however, take in the order of days to train. Optimizing hyper-parameters is another factor that adds to the training time. This thesis aims to analyze the training pattern on Convolutional Neural Networks from a systems perspective. We perform a thorough study on the effects of systems resources like DRAM, persistent storage (SSD/HDD space), and GPU on the training time. We explore how one could avoid bottlenecks in the data processing pipeline in the training phase. Our analysis illustrates how GPU utilization can be maximized in the training pipeline by choosing the right combination of two hyper-parameters - batch size and the number of data prefetching worker processes. We also take a step forward and propose a novel strategy to optimize these hyper-parameters by estimating the

maximum batch size that can be used. Additionally, our strategy provides an approximate efficient combination of batch size and the number of worker processes for the given resources.

Table of Contents

Acknowledgments **v**

Abstract **vii**

List of Tables **xi**

List of Figures **xiii**

Chapter 1. Introduction **1**

Chapter 2. Background **6**

2.1 Deep Learning 6

2.1.1 Artificial Neural Networks 6

2.1.2 Deep Neural Networks 7

2.1.2.1 Convolutional Neural Networks 10

2.1.2.2 Recurrent Neural Networks 10

2.2 Hyper Parameter Optimization 11

2.3 Training of Deep Learning Models 11

2.3.1 Frameworks 11

2.3.1.1 PyTorch 12

2.3.2 Training Process 12

2.3.3 Training of model on PyTorch - System specifics 12

2.3.3.1 Persistent storage SSD/HDD 13

2.3.3.2 Worker Processes 13

2.3.3.3 Host DRAM 14

2.3.3.4 GPU 14

2.3.3.5 Number of GPUs 15

Chapter 3. Analysis **16**
3.1 Experimental Setup 16
3.2 Batch size 18
3.3 Number of pre-fetching workers 21
3.4 Number of GPUs 24
3.5 DRAM size 24
3.6 Results 25

Chapter 4. Estimation of batch size and workers - Methodology **27**
4.1 Maximum Batch Size 28
4.2 Number of Workers 32

Chapter 5. Evaluation **36**
5.1 Experimental Setup 37
5.2 Maximum Batch Size 37
5.3 Workers and Batch size estimate 40
5.4 CNN model performance measure 43
5.5 Manual Search vs Algorithm 45
5.5.1 Manual Search 46
5.5.2 Algorithm 47
5.5.3 Comparison 47

Chapter 6. Limitations **50**

Chapter 7. Conclusion **52**

Bibliography **54**

Vita **62**

List of Tables

3.1 **Maximum number of workers for a DRAM size.** The table shows maximum number of workers feasible on a particular DRAM and fixed batch size. 25

5.1 **Estimated and actual maximum batch size for different CNN models.** The table shows the actual maximum batch size possible, estimated number for maximum batch size and the percentage of error between them for 6 different CNN models. 39

5.2 **Estimated maximum batch size for 2 different GPU memory sizes.** The table illustrates the estimation of maximum batch size dependence on GPU memory. The maximum batch size is smaller for smaller GPU memory. 40

5.3 **Resnet50 - training time per epoch and avg. GPU utilization for set of workers and batch sizes.** The table shows the training time epoch and average GPU utilization in case of estimated numbers(bold) from the algorithm performed optimally as compared to other number of workers and batch sizes for model Resnet50. 42

5.4 **Conv3d - training time per epoch and avg. GPU utilization for set of workers and batch sizes.** The table shows the training time epoch and average GPU utilization in case of estimated numbers(bold) from the algorithm performed close to optimal as compared to other number of workers and batch sizes for model Conv3d resnet. 43

5.5 **Resnet34 - training time per epoch and avg. GPU utilization for set of workers and batch sizes.** The table shows the training time epoch and average GPU utilization in case of estimated numbers(bold) from the algorithm performed a close to optimal as compared to other number of workers and batch sizes for model Resnet34. 44

5.6 **Variation of estimated numbers of workers and batch size on DRAM size.** The table shows the change in the estimated number of workers and batch size as the DRAM size is different. The workers are more for larger DRAM size of 128 GB. 44

5.7 **Comparison of validation accuracy and validation loss for estimated numbers by algorithm with default numbers of workers and batch size.** The table shows that estimated numbers performed well with respect to validation accuracy and loss as compared to default numbers for two different models when ran for same number of epochs. 45

5.8 **Comparison of time taken to search for batch size in case of Manual search vs Algorithm.** The table shows that the algorithm is faster or comparable to reach to batch size estimate as compared to manual search for different models. 48

List of Figures

2.1 **Simple Neural Network** 8

2.2 **Forward and Backward propagation** 8

2.3 **Deep Neural Network** 9

3.1 **Training time at each epoch.** The figure shows that training time over epochs does not vary much. 18

3.2 **Batch size variation.** The figure shows decrease in training time per epoch as increase in batch size from 32 to 256 and increase in GPU utilization and Memory utilization till batch size 256 given fixed number of workers and fixed number of GPUs. 19

3.3 **Maximum GPU utilization vs pair of workers and batch size.** The figure shows that a certain batch size attain maximum GPU utilization on all 4 GPUs as the number of workers are changed. Batch size 128 has maximum GPU utilization when workers are 4(fewer) as compared to 512 when workers are 32(more). 20

3.4 **GPU Memory utilization vs pair of workers and batch size.** The figure shows that GPU memory utilization increases on all 4 GPUs as the batch size is varied from 32 to 512. The pattern is same even when the number of workers are changed from 4 to 32. 22

3.5 **Number of workers variation.** The figure shows decrease in training time per epoch as increase in number of workers from 0 to 32. GPU utilization increases as workers are increased and Memory utilization is similar for all number of workers given fixed batch size and number of GPUs. 23

3.6 **Max GPU utilization value different for different batch size.** The figure illustrates that the pattern of GPU utilization remains same over the number of workers for different batch sizes but the maximum GPU utilization value differs. The maximum GPU utilization attained is 83% in case of batch size 32 while 96% in case of batch size 256. 23

3.7 **Number of GPUs variation.** The figure shows that as the number of GPUs increases from 1 to 4, the training time per epoch, GPU utilization and GPU memory utilization deceases given fixed batch size and number of workers. The pattern was same on all GPU(s), the figure is shown for one of them, GPU:0. 24

3.8 **Training time per epoch for 2 different DRAM sizes.** The figure illustrates that training time per epoch were similar for different 2 DRAM sizes, 20 GB and 128 GB. The training time per epoch is shown for 3 different batch sizes given fixed number of workers and fixed number of GPUs. 26

Chapter 1

Introduction

Machine Learning (ML) is prominent today, and is widely used to solve computational problems in several fields including medicine[1][2], autonomous robots[3][4], games[5], etc. There are a multitude of techniques in ML to solve diverse problems; for example, Hidden Markov Models[6], sequence labeling in NLP[7][8], Reinforcement Learning[9][10], etc. One such prominent techniques is Deep Learning (or Deep Neural Networks) [11].

Deep Neural Networks (DNNs) learn complex relations between the input and the output and is proven to be effective in predicting unseen data [12][13]. However, DNNs require several GB of training data for the best predictions. These networks also perform a lot of computation and require extensive usage of resources like persistent storage (SSD/HDD), DRAM and GPUs. The time to train these networks depends on the effective usage of available resources and the size of training data. GPUs in particular, play a significant role in training such networks [14]. We suspect that the training time of these networks can be fairly reduced if GPU is utilized efficiently.

Training DNNs involves fetching data from the disk (or memory) and feeding it to the GPUs for computation. Additionally, to build a DNN model, some parameters are fixed before they are trained on the data, also known as hyper-parameters [15]. Performance of a model significantly depends on the values of these hyper-parameters and needs to be chosen carefully.

The number of samples to work through before updating the internal model parameters is termed the *batch size*. To keep the GPU fully utilized in the critical path, DNN frameworks like PyTorch use multiple processes called *workers*, which retrieve batches from disk in parallel and keep it ready for GPUs. Batch size and the number of workers are two important hyper-parameters that dominate the performance of a DNN model.

The optimal batch size and the number of workers are chosen by trial and error today. Researchers manually pick values of this hyper-parameters[16], which often times may not be the best choice across different hardware configurations of machines on which the model could be trained. Every time the hardware configuration of the machine changes, the appropriate hyper-parameters must be chosen again. Such manual search takes up a significant amount of time and there has been no such efficient way to automate this process. The domain of the search space of these hyper-parameters also varies with the DNN model, host DRAM capacity, GPU memory, and others.

This thesis analyzes the training bottlenecks in a specific class of DNN models, called the Convolutional Neural Networks (CNN), in terms of the total training time and GPU utilization. CNNs are widely used for computer vision problems like image classification, and video activity recognition with typical dataset size of 100s of GBs like Imagenet(140 GB) and takes days to train the network [12]. Our analysis reveals that a certain combination of batch size and workers result in lower training time and higher GPU utilization.

Our work proposes a novel methodology to compute an approximate estimate for the maximum batch size and the number of workers for given batch size. Understanding the maximum permissible batch size allows researchers to know the domain of batch sizes to experiment with, beforehand. This approach also provides an estimate of the optimal combination of batch size and number of workers to keep the GPU fully utilized. We expect these estimates to be useful to the ML community to know upfront, avoiding massive manual effort. We test our strategy, on the Imagenet dataset on PyTorch. We evaluate the effectiveness of our strategy by comparing how close the estimated numbers are to the manually chosen numbers, and how the estimated parameters affect validation accuracy and loss.

The estimates we predict take into account the hardware configurations and resources available on the machine. For instance, the estimate for optimal

batch size is 160 for a system with 16 GB GPU memory, whereas it is 100 for an 11 GB GPU memory system. Interestingly, the estimates predicted by our approach are not typical choices made by researchers during their manual search. For example, in the case of Alexnet[17], the maximum batch size we find is 1300 in an 11 GB GPU memory system. Such large batch sizes are not even a part of the experimentation in a manual setting, and the default value chosen is 128. Therefore, our approach helps to explore unorthodox values for batch size, maximizing GPU utilization as much as possible.

However, the typical assumption is that large batch sizes suffer from the problem of convergence. Larger the batch size, lesser the number of updates in an epoch, thereby delaying model convergence. But, prior work explores the impact of large batch size on convergence [18], and show that large batch sizes of the order of 8192 can still converge in about an hour [19] in case of Imagenet.

Our work is not without limitations. Our approach is specific to CNNs as they are data intensive in the training phase. Our estimation strategy is specific to CNNs and extending it to other neural networks would require careful examination of the impacts of specific memory units like LSTMs for instance. Furthermore, the analysis that led to our estimation strategy is based on the Pytorch framework. There are several other frameworks like Tensorflow, MxNet, etc, which provide appropriate abstractions for building DNNs, each with its own specification for fetching and training data on GPUs.

However, our preliminary experiments show similar results on TensorFlow, and we expect that it would be similar over other platforms.

Chapter 2

Background

This chapter provides background on Deep Learning and System components. It first describes what are artificial neural networks followed by deep learning and explanation on hyperparameter optimization. Later it covers the different system components that play a role in the training of these deep learning models.

2.1 Deep Learning

2.1.1 Artificial Neural Networks

"Artificial neural networks (ANN) or connectionist systems are computing systems vaguely inspired by the biological neural networks that constitute animal brains. The neural network itself is not an algorithm, but rather a framework for many different machine learning algorithms to work together and process complex data inputs. Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules"[20].

These ANNs are comprised of small computing units known as percep-

trons[21] which given certain inputs, perform function computation on those inputs and generates outputs. These perceptrons are arranged and connected in a different manner constructs a neural network as shown in figure 2.1. The neural networks have specific input layers which have the input, the hidden layers which does computations and the output layer which generates the output. The transfer of information from the input layer to the output layer is called forward propagation. Every perceptron has a given activation function which triggers output to be sent to the next layers. Once the output is generated it is compared from the required output and the error is then propagated backward from the output layer till the first hidden layer, this is called backpropagation[22] and the perceptrons learn the weight of their gradient function. This is shown in the figure 2.2.

Each perceptron tries to reach a convergence point and thus the whole neural network tries to approach convergence. The convergence is when the neural network when accuracy on validation/test data starts decreasing or when the error difference between the estimated output and expected output is not changing much.

2.1.2 Deep Neural Networks

There can be as many hidden layers in a neural network. When the hidden layers are many it is called a deep neural network[23] as shown in figure 2.3.

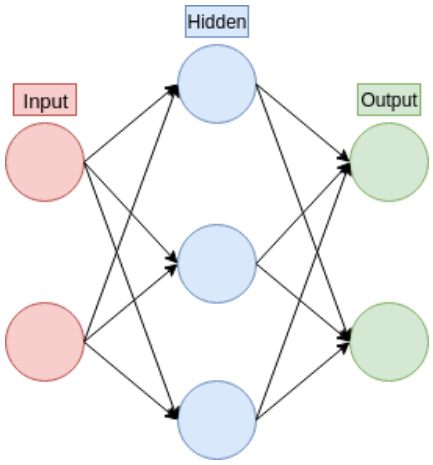


Figure 2.1: **Simple Neural Network**

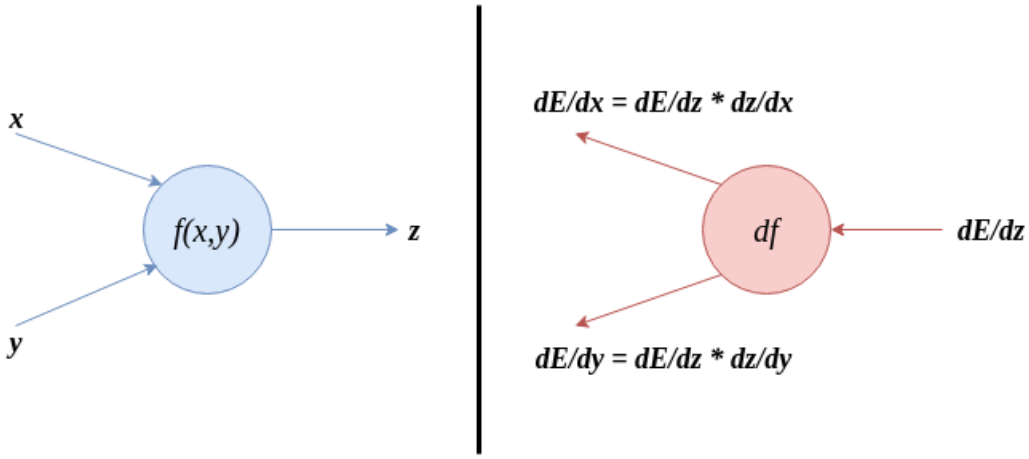


Figure 2.2: **Forward and Backward propagation**

When each neuron is connected to all the other neurons in the consecutive layers it is known as feedforward networks[24] as in the figure 2.3. These connections can be changed and a perceptron might not be connected to all the other perceptrons. Each layer can also have separate activation and computation function. Each such combination of connection and functions can

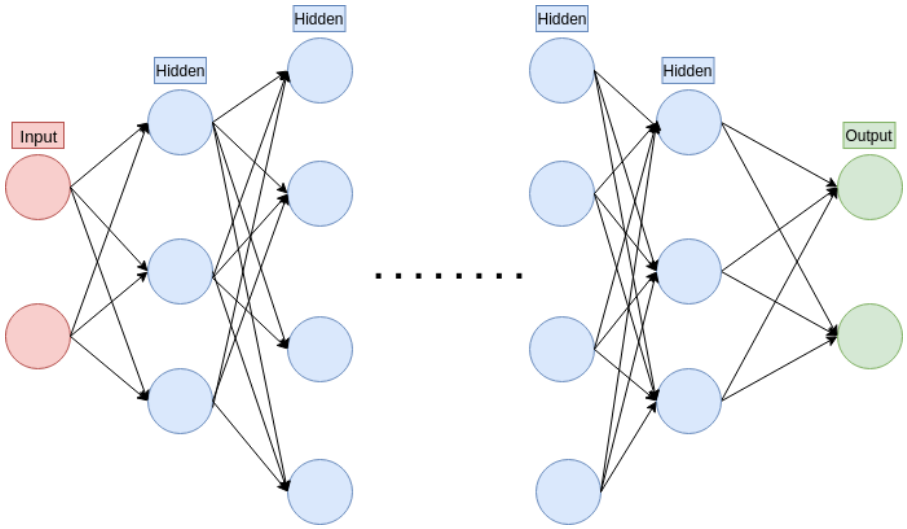


Figure 2.3: **Deep Neural Network**

result in a different neural network.

There are deep neural networks which are developed specifically to the applications. For example, Convolutional Neural Networks constituted of convolution layers, max-pooling layers and so on, are explained in detail in the following section. These are predominantly used for image classification and vision problems. Recurrent Neural Networks are another class of deep neural networks which are constituted of memory components known as Long Short Term Memory(LSTM) layers and recursive layers which are useful for problems in Natural Language Processing.

2.1.2.1 Convolutional Neural Networks

A Convolutional Neural Network (CNN)[12] is a Deep Learning algorithm which takes an input image, process the image, learn the weights and biases and tries to assign importance to various aspects/objects in the image and be able to differentiate one from the other.

A CNN, through the application of relevant filters, is able to capture the Spatial and Temporal dependencies in an image. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. Each layer does a different operation on an image and tries to reduce the parameters and only keep the ones which are relevant. It has a different type of layers, for example, convolution layers, max-pooling layers, fully connected and at the end output average layer like softmax. [25]

2.1.2.2 Recurrent Neural Networks

A recurrent neural network (RNN) [26] is a class of artificial neural network where connections between nodes form a directed graph along a temporal sequence. Each node has input as output from the previous time step, thus it can learn temporal information.

2.2 Hyper Parameter Optimization

In machine learning, a hyperparameter is a parameter whose value is fixed before the training process starts instead of one whose values is derived over the training process. Choosing these fixed values is of importance as this decides the performance of a model in terms of accuracy with respect to a problem. The process of choosing these values is called Hyperparameter Optimization[27].

There are different ways to do hyperparameter optimization. Some common ways are Grid search, random search[28], Bayesian methods[29]. Grid search is a common one where one manually go over some discrete values which require prior knowledge of acceptable values and suffers from the curse of dimensionality.

Some common hyperparameters to learn in Machine learning are the number of hidden layers, learning rate, batch size, activation functions.

2.3 Training of Deep Learning Models

2.3.1 Frameworks

Deep Learning Frameworks are developed to provides the abstraction for to write machine learning models, which helps in building neural networks and encapsulating GPUs and performing parallel processing via cuda. Each

framework is built in a different manner for different purposes. Some common ones are, Tensorflow[30], PyTorch[31], Caffe[32], Keras[33], MXNet[34].

2.3.1.1 PyTorch

We chose PyTorch for our experimentation because of its wide usage and easy adaptation.

2.3.2 Training Process

The training process of a machine learning model on an abstract level involves the following steps:

1. Preporceesing training data(optional) - This involves doing computation on data before starting the training. For example, resizing images.
2. Preparing data - Getting data from disk.
3. Feeding data to GPUs iteratively in a chunk of batch sizes till covered the whole data.
4. Training on whole data multiple times.
5. Test on validation/test data

2.3.3 Training of model on PyTorch - System specifics

Training on PyTorch involves preprocessing the data and load it via loaders provided by PyTorch which might be specific to the type of data. The

most general one is the `DataLoader`. Once the data is prepared, the model is given the whole data multiple numbers of times to train on, these runs are known as epochs. In all epochs, the data is divided into multiple chunks of batch size amount and the model is given 1 batch at a time to train on, also known as iterations.

2.3.3.1 Persistent storage SSD/HDD

The dataset resides in SSD/HDD, PyTorch fetches the data from it and preprocess the images and store it as a list of tensors with the value of the class label too. Each list item represents images as tensors. Each tensor in the list is around 1 MB in size and as the number of images is around 1 million the overall size of the list surpass the DRAM. Thus, getting the data from the list can result in IO from disk as all the images cannot reside in the memory. This IO might delay the data fetching process and thus increase the overall training time for a model.

2.3.3.2 Worker Processes

To overcome some of the IO occurrences, PyTorch initializes some workers which solely performs prefetching data for the next iterations in the background. PyTorch uses a class called DataLoader[35], this initiates a given amount of workers(processes), these workers run in the background and get the tensors at a given index from the list forming a batch and put it in a multiprocessing queue[36].

The main worker runs a loop wherein each iteration gets the batch size amount of data from the queue and sends the data to the GPU devices. This for loop runs for all the epochs. The background workers encounter the latency due to IO if any. Thus, there is a need for an optimal amount of workers such that they are sufficient enough to hide the disk IO happening while fetching the data. Though the workers cannot be infinite, each of them fetches batch size amount of data and keep in memory and thus limited by the amount of host DRAM.

2.3.3.3 Host DRAM

The DRAM size is used to cache the data and storing the required workspace for the process. The more the DRAM the more will be the caching and thus fewer chances of page faults followed by disk IO. The DRAM limits the number of workers as well.

2.3.3.4 GPU

GPUs are a vital component in training of machine learning models. GPUs do parallel operations and make the training process faster. The GPU utilization does depend on the batch size, the amount of data it is processing at a time. The Batch size is the number of samples model trains on at a time and then update the parameters of the model. Size of a batch needs to be optimal as if it small enough it will not utilize the GPUs fully. It will require

a large amount of time to finish the training as there will be more number of iterations in an epoch. On the other hand, if the batch is very big, it will not be able to reside in the GPU memory and will not be able to run.

2.3.3.5 Number of GPUs

The model training can happen on multiple GPUs as well. PyTorch in case of multiple GPUs divides a batch size by the number of GPUs and sends the corresponding data to each of them. One of the GPU acts as a coordinator and sends the data to other GPUs and gets back the updated weight from them and updates the gradients. The advantage of having multiple GPUs is that we can have a larger batch size that gets distributed within the GPUs. Thus the optimal value needed is the batch size per GPU.

Chapter 3

Analysis

In this chapter, we performed several experiments to analyze how the following factors affect the training of a model, resource utilization and how the storage bottlenecks can be removed from the training pipeline.

- Number of prefetching workers
- Batch size
- Number of GPUs
- DRAM size

3.1 Experimental Setup

To analyze the effect of these factors, several training runs were done on common neural network model Resnet50. The model was trained on the problem of image classification with ImageNet as training dataset of size 140GB with total 1000 classes. The Resnet50 code used is the official code provided by PyTorch[37]. The runs were primarily done on a machine with Four NVIDIA 16GB Tesla V100 SMX2 GPUs, Two 960 GB 6G SATA SSD, 128GB ECC

Memory and Two Intel Xeon E5-2667 8-core CPUs at 3.20 GHz.

The batch size was discretely varied among 32, 128, 256, 512. The number of workers was also discretely varied among 0, 4, 8, 16, 32 where 0 represents no workers, no prefetching. Number of GPUs taken into account were 1, 2, 3, 4. The DRAM size considered was 20 GB comparable to GPU memory and 128 GB comparable to dataset size 140 GB.

In training of a model, the first epoch takes a small amount of extra time than the other epochs because of initial initialization, preprocessing images, initializing workers, etc. After the first epoch, the second epoch onwards, all the epochs show similar behavior in terms of time, utilization and memory consumption. The plot in figure 3.1 shows the time behavior over the first 30 epochs. As the behavior of the rest of the epochs except first were similar, average data for over 3 epochs were considered for analysis.

The runs are examined on the basis of time taken to finish the training per epoch, average GPU utilization and average and maximum memory utilization over 3 epochs.

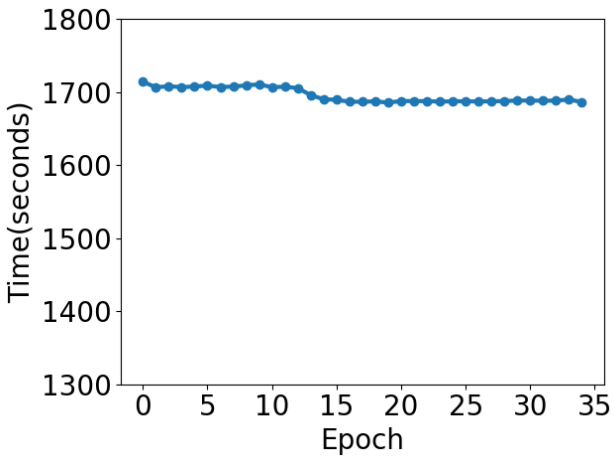
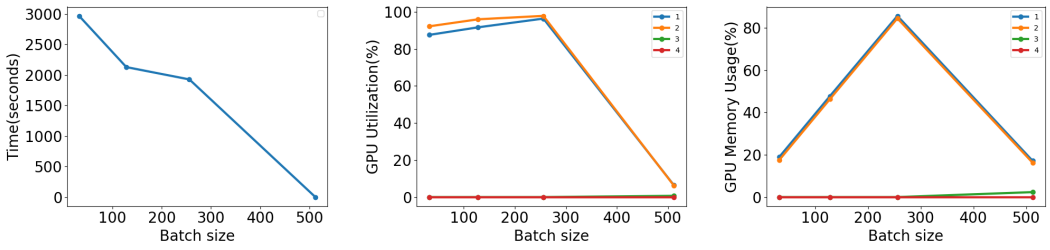


Figure 3.1: **Training time at each epoch.** The figure shows that training time over epochs does not vary much.

3.2 Batch size

The batch size is varied over 4 different values. The case when the number of workers was 16, the number of GPUs were 2 and the batch was varied is shown in figure 3.2. We can see from the figure that as we increase the batch size from 32 to 256, the average time taken per epoch is getting reduced, whereas GPU Utilization increases reach close to 95% at 256 and Memory Utilization also increasing reaches to maximum 80% at 256. But as increased more to 512, it ran out of memory as each GPU had 256 size batch.

One more factor to notice is, as shown in figure 3.3 as we increase the number of workers and see the utilization pattern over batch size when the workers were less the lesser batch size had better utilization and as the workers increased the maximum GPU utilization was higher for higher batch sizes.



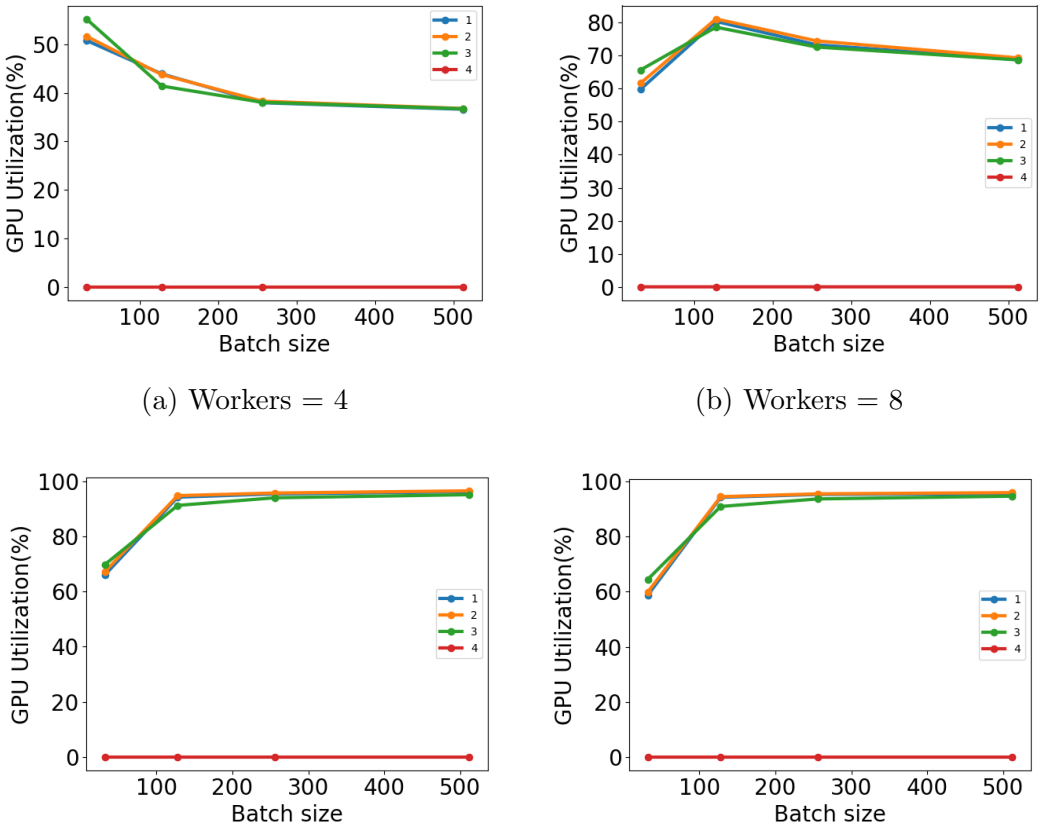
(a) Time (b) GPU Utilization (c) Memory Utilization

Figure 3.2: **Batch size variation.** The figure shows decrease in training time per epoch as increase in batch size from 32 to 256 and increase in GPU utilization and Memory utilization till batch size 256 given fixed number of workers and fixed number of GPUs.

The reason could be that there has to be an optimal balance between getting data and processing data. So when there are fewer workers, as higher batch size takes more time in IO, the time to fetch the data is more so the workers are not enough to keep the GPUs occupied. So each batch size requires a certain number of workers to get better GPU utilization and less training time.

The above reason can easily be concluded from the 4 plots in figure 3.3, which shows the shift in maximum GPU Utilization as workers increases and varied over different batch sizes.

However, a similar pattern is not observed for Memory Utilization, it is always high for higher batch size as shown in figure 3.4, that is because PyTorch does preallocation in GPU memory which is higher for higher batch size.



(a) Workers = 4

(b) Workers = 8

(c) Workers = 16

(d) Workers = 32

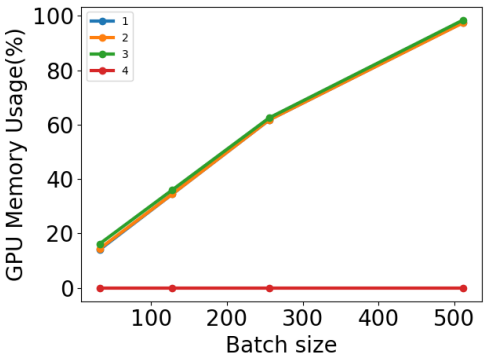
Figure 3.3: **Maximum GPU utilization vs pair of workers and batch size.** The figure shows that a certain batch size attain maximum GPU utilization on all 4 GPUs as the number of workers are changed. Batch size 128 has maximum GPU utilization when workers are 4(fewer) as compared to 512 when workers are 32(more).

3.3 Number of pre-fetching workers

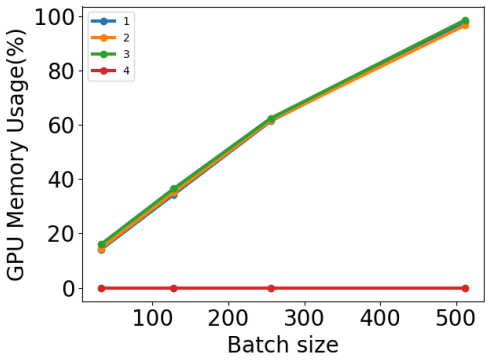
Workers do the work of prefetching data that is going to be accessed. The more the workers, the more will be the prefetching. But the number of workers is limited by the DRAM size.

Shown in figure 3.5, as the number of workers, increases the average time per epoch reduces, the GPU utilization increases. However, as PyTorch does pre-allocation according to the batch size, the number of workers doesn't have much effect on memory utilization and is all similar overall number of workers.

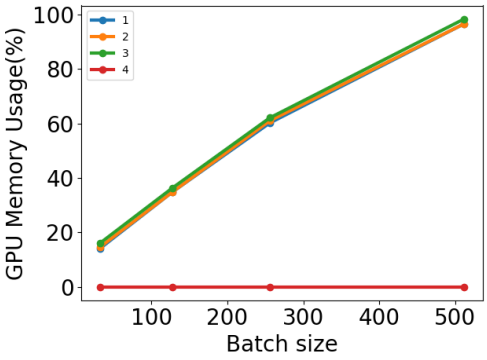
The number of workers has the same pattern on GPU Utilization but one thing that is different is the range. The workers help more when the batch size high as the number of time to get data complements the time to process the data. As shown in figure 3.6, in case of batch_size=32 the utilization range from 17% to 83% but in case of batch_size=256 it ranges from 13% to 97%.



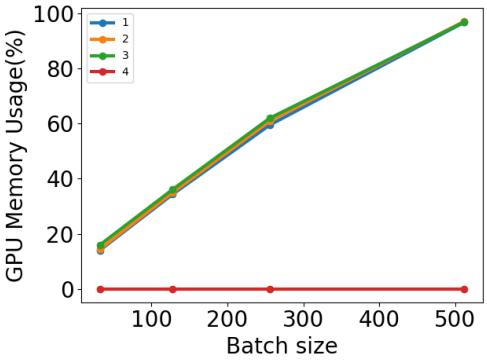
(a) Workers = 4



(b) Workers = 8

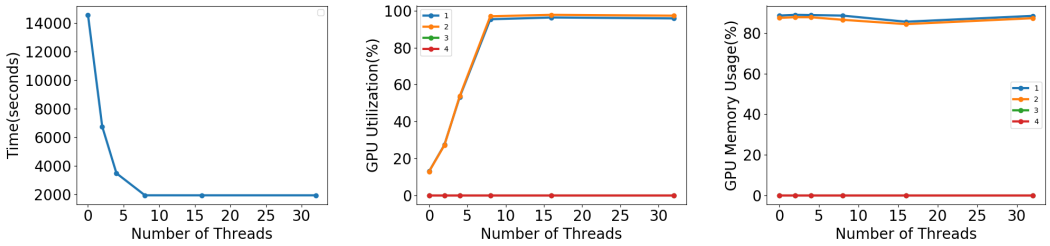


(c) Workers = 16



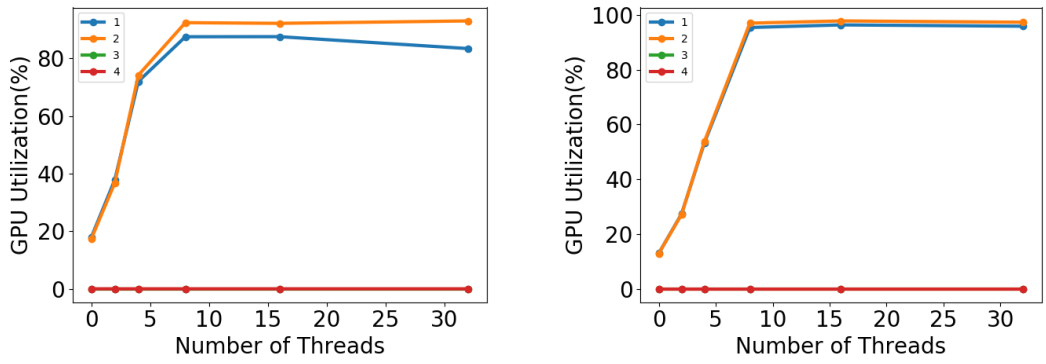
(d) Workers = 32

Figure 3.4: **GPU Memory utilization vs pair of workers and batch size.** The figure shows that GPU memory utilization increases on all 4 GPUs as the batch size is varied from 32 to 512. The pattern is same even when the number of workers are changed from 4 to 32.



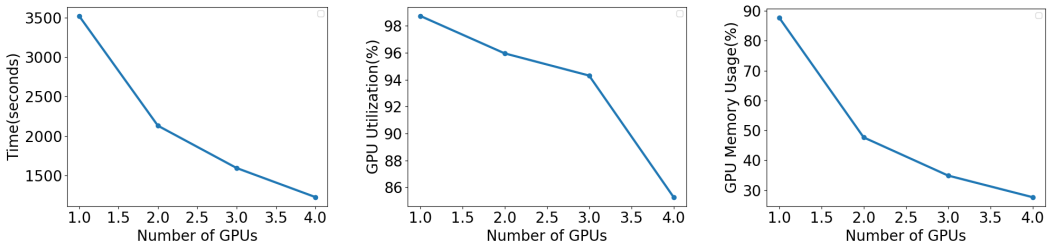
(a) Time (b) GPU Utilization (c) Memory Utilization

Figure 3.5: **Number of workers variation.** The figure shows decrease in training time per epoch as increase in number of workers from 0 to 32. GPU utilization increases as workers are increased and Memory utilization is similar for all number of workers given fixed batch size and number of GPUs.



(a) batch_size=32 (b) batch_size=256

Figure 3.6: **Max GPU utilization value different for different batch size.** The figure illustrates that the pattern of GPU utilization remains same over the number of workers for different batch sizes but the maximum GPU utilization value differs. The maximum GPU utilization attained is 83% in case of batch size 32 while 96% in case of batch size 256.



(a) Time (b) GPU Utilization (c) Memory Utilization

Figure 3.7: **Number of GPUs variation.** The figure shows that as the number of GPUs increases from 1 to 4, the training time per epoch, GPU utilization and GPU memory utilization deceases given fixed batch size and number of workers. The pattern was same on all GPU(s), the figure is shown for one of them, GPU:0.

3.4 Number of GPUs

The role of multiple GPU is more amount of parallel processing. The more the number of GPUs will be, the more can be the batch size and faster the training explained in section 2.3.3.5.

As shown from figure 3.7, as the number of GPU increases, for a given batch_size and number of workers, the training time per epoch reduces, the utilization on the other hand reduces as well because as the number of GPU increases, batch size per GPU decrease and thus does not keep the GPU utilized. Similarly, the memory utilization decreases too.

3.5 DRAM size

The DRAM size limits the number of workers as shown in table 3.1. The run was done on 2 GPUs. The number of workers is picked from the set

0, 2, 4, 8, 16, 32, 64 and reported the maximum among them.

Batch size	Max workers ‖20 GB	Max workers ‖128 GB
128	32	32
256	16	32

Table 3.1: **Maximum number of workers for a DRAM size.** The table shows maximum number of workers feasible on a particular DRAM and fixed batch size.

However, DRAM size did not affect the training time significantly given the fact that more DRAM could cache more data. As shown in figure 3.8 when the model was run for multiple batch sizes and 32 workers on two different DRAM sizes, the training time per epoch were similar.

3.6 Results

As we saw in the section 3.2 the shift of optimal batch size with respect to the number of workers. From the runs, it is evident that the training time and efficient GPU Utilization is dependent on the optimal combination of batch size per GPU and number of workers given the number of total GPUs and DRAM size.

In the current time, there exists no such direct way to get the optimal combination of batch size and the number of workers. ML engineers tend to

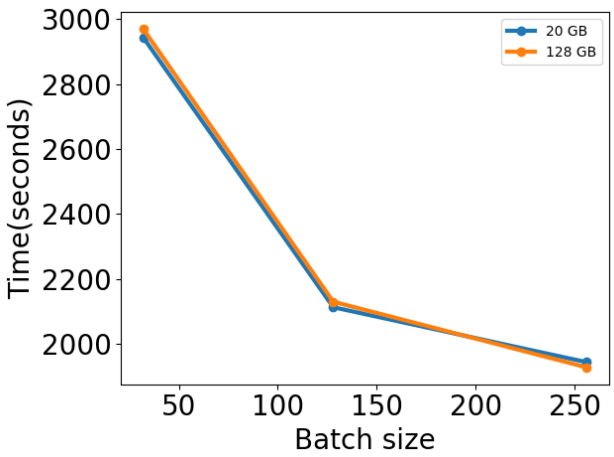


Figure 3.8: **Training time per epoch for 2 different DRAM sizes.** The figure illustrates that training time per epoch were similar for different 2 DRAM sizes, 20 GB and 128 GB. The training time per epoch is shown for 3 different batch sizes given fixed number of workers and fixed number of GPUs.

manually search for the combination and do the training. In the next chapter, we put forward a methodology to estimate a maximum batch size which would reduce the search space and also an approximate number for optimal values of workers and batch size. Our constraints are also useful to estimate the number of workers given the batch size.

Chapter 4

Estimation of batch size and workers - Methodology

This chapter we will explain the methodology to estimate approximate optimal batch size and number of workers. As we concluded in the previous section that there is a need to know the batch size and number of workers for lesser training time and better resource utilization. But what range should be explored for batch size and what would be the efficient amount of workers that would work for given batch size? And as we saw from the analysis in previous chapter 3, in the case of Imagenet dataset[38] higher batch size helped in better resource utilization.

Here we are trying to estimate the range of batch size which could be used to train a given convolutional neural network. The maximum value of batch size that could be used is limited by the GPU memory. As the batch size increases, the GPU memory requirement increases. The GPU memory is also consumed by the model and the model processing metadata, the whole memory could not be used to just keep the input batch size amount of data. Therefore, the maximum batch size possible could not be directly computed

from the available GPU memory.

4.1 Maximum Batch Size

To find the maximum batch size, we need a way to calculate the GPU memory usage. This will help in estimating how much input data can be put alongside and thus the batch size.

In the case of convolutional neural networks, there are multiple layers and each layer has its own parameters. Let's assume each i^{th} layer has total n_i parameters, these parameters could be anything like weights particular to the layer i. And each parameter has some p_i size in bytes. Thus, the amount of memory used by these parameters for a layer i would be, $n_i * p_i$.

The input images of size $H * W * D$ will go inside the first layer, the first layer thus will get tensors of total size $B * H * W * D$ where B is the batch size. The first layer will further generate output of size $h_1 * w_1 * d_1$ for each image and thus output tensor of size $B * h_1 * w_1 * d_1$. This output will go into the next layer and so on and so forth. Thus a layer i will generate output of size $B * h_i * w_i * d_i$. These outputs of each layer are also known as feature maps and are stored in the GPU memory, they are needed to be kept in memory to calculate the gradients at the time of backpropagation. The gradients when computed have a similar size as the feature map and thus also use a similar amount of memory.

Thus, the GPU memory usage for a batch can be estimated as:

$$M = B * I + B * O + \sum_{i=1}^{L} n_i * p_i + B * O_i + B * O_i \tag{4.1}$$

where M is the GPU memory usage, I is the size of an input image tensor, O is the size of an output image tensor, L is the total number of layers and for each layer we have memory consumption by parameters as $n_i * p_i$, by feature maps as B * O_i and by each gradient as B * O_i. Here O_i is equivalent to $h_i * w_i * d_i$.

Memory consumed by parameters and output of each layer is hard to get due to lack of performance counters in the GPUs. Thus to estimate an upper bound on the memory usage we will assume

$$N = \max_{1 \leq i \leq L} n_i \tag{4.2}$$

$$P = \max_{1 \leq i \leq L} p_i \tag{4.3}$$

$$O_M = \max_{1 \leq i \leq L} O_i \tag{4.4}$$

replacing the values of n_i, p_i and O_i to their maximum values from equation 5.2 and 5.3 in equation 5.1, we get:

$$M_u = B * I + B * O + \sum_{i=1}^{L} N * P + B * O_M + B * O_M \tag{4.5}$$

which can be written as:

$$M_u = B * I + B * O + L * N * P + 2 * L * B * O_M \tag{4.6}$$

where M_u represents the upper bound on GPU memory usage.

Replacing $X = L * N * P$ and $Y = L * O_M$ in equation 5.6, we get:

$$M_u = B * I + B * O + X + 2 * B * Y \tag{4.7}$$

Equation 5.7 is now an equation in two variables X and Y.

To find the maximum batch size, we need two data points for our equation 5.7 to find the value of the two variables X and Y. To get the data points, we can run a model for some 15-20 iterations at two different batch sizes lets say b_1 and b_2. The memory usage can be found by `torch.cuda` memory management apis [39] namely `torch.cuda.memory_allocated` and `torch.cuda.memory_cached`. The input tensors size can be found using `input.nelements*input.element_size` and similarly for size of output tensors using `output.nelements*output.element_size`. The memory usage for a batch b is the average over the iterations it ran for.

1. Memory usage M_1 for b_1

2. Memory usage M_2 for b_2

3. Input size per tensor I = $\frac{I_1}{b_1}$, I_1 is the input size for b_1

4. Input size per tensor O = $\frac{O_1}{b_1}$, O_1 is the output size for b_1

5. Solve for X and Y in equation

   $M_1 = b_1 * I + b_1 * O + X + 2 * b_1 * Y$

   $M_2 = b_2 * I + b_2 * O + X + 2 * b_2 * Y$

Once we get X and Y from the above procedure, to find the maximum batch size, we will put the available GPU memory, GPU_M as M_u and use the same I and O which we found above. Thus, the maximum value of batch size, B_m can be found by:

$$GPU_M = B_m * I + B_m * O + X + 2 * B_m * Y \tag{4.8}$$

$$B_m = \frac{GPU_M - X}{I + O + 2 * Y} \tag{4.9}$$

The maximum GPU memory available, GPU_M should be given as 1-1.5 GB less than actual to handle the error for over-approximation of the upper bounds on X and Y.

This maximum batch size is the upper bound on the range for the batch size on which the model could be trained given the GPU resource. For

to estimate maximum batch size in case of multiple GPUs, they can be easily computed by multiplying the count of GPUs with the B_m.

4.2 Number of Workers

As the mere batch size does not decide for the optimal training time and GPU Utilization, we need to find the corresponding number of workers as well. The workers are the ones due to which storage bottlenecks are avoided in the training pipeline.

The workers are constrained by the DRAM size. As the worker's prefetch the data in the amount of batch size. So if there are W workers and B batch size then

$$W * B * I < DRAM \tag{4.10}$$

where I is the input size of one tensor.

Another constraint on workers is that there should be a minimum amount of workers such that GPUs are never bottlenecked by the data. The workers fetch data for GPUs, let's say the time to fetch batch size amount of data by 1 worker is t_d and the time to process a batch by 1 GPU is t_g. So if we have W workers, then in t_d amount of time we would pre-fetch W*B amount of data. Thus the GPU can be kept busy for another W rounds. In

the meantime, when the GPUs are busy with processing these W batches, the workers should get more data for the next rounds. Thus, we want our time to fetch data t_d to be less than the time to process a batch by GPU multiplied by workers, that is:

$$t_d <= t_g * W \tag{4.11}$$

One constraint on batch size is that it should be less than the maximum batch size we computed in the previous section.

$$B <= B_m \tag{4.12}$$

Thus, to find the feasible solution for the number of workers and batch size given the above constraints, the problem is framed as a Lagrange multiplier[40] problem where we are trying to maximize the batch size given the constraint functions. The idea behind trying to maximize batch size is that, as we want to keep GPU fully utilized, so the more the batch size under constraint(to minimize time) the better. At the same time, we want to keep the number of workers as enough to attain minimal time but at the same time not too many to waste CPU cycles.

$$\text{maximize } B \tag{4.13}$$

$$W * B * I < DRAM \tag{4.14}$$

$$t_d <= t_g * W \tag{4.15}$$

$$B <= B_m \tag{4.16}$$

The above equation has two variables W and B which we are trying to optimize. To solve the above equations, the value of t_d and t_g will be needed. Both t_d and t_g are dependent on the machine and batch size. But after doing several experimentations with different batch sizes on a model, it could be said that empirically the ratio of t_d and t_g doesn't vary over different batches. Given a room for certain error, we can get the constant value for constraint in equation 5.15.

Thus to compute the values of t_d and t_g, one needs to run the model for 50 iterations and average the value for GPU computation and Data fetch time. This could be done for two different batch sizes and the constraint could be the maximum value of $\frac{t_d}{t_g}$ over the two batches. The B_m can be computed using the method mentioned in the previous section needed for constraint in equation 5.16.

The above method to estimate the number of workers could easily be extended for multiple GPUs. In case of multiple GPUs N if the batch size is B, the time to process B batch size will become t_g divided N. Here, t_g is again

time to process batch B at 1 GPU. Thus the constraint in equation 4.15 will translate as:

$$t_d <= (\frac{t_g}{N}) * W \tag{4.17}$$

As a corollary to the results, the constraint in equation 4.14 is also sufficient to find out the number of workers given batch size.

Chapter 5

Evaluation

This chapter evaluates the proposed methodologies in two categories, the correctness and the usefulness for the same. It answers the following questions:

- How accurately is the maximum batch size computed? (section 5.2 For what maximum batch size the model ran?)
- How accurately the number of workers and batch size are estimated? (section 5.3 For what numbers of workers and batch size model performed best?)
- How beneficial the numbers are in terms of CNN training? (section 5.4 How model performed in terms of accuracy and loss for these numbers?)
- How beneficial the method is as compared to manual search with respect to time? (section 5.5 Compares the two approaches in terms of time takes to find batch size)

5.1 Experimental Setup

The experiments were run on two different machines, M1 with 32 GB DRAM, two GTX P0 GPUs, and 1 TB HDD and M2 with Four NVIDIA 16GB Tesla V100 SMX2 GPUs, Two 960 GB 6G SATA SSD, 128GB ECC Memory and Two Intel Xeon E5-2667 8-core CPUs at 3.20 GHz.

Experiments were performed on different convolutional neural networks namely resnet34, resnet50, resnet152, alexnet, 3d convolutional resnet neural networks for video recognition. The datasets used for CNN model training were ImageNet dataset[38] and UCF101 dataset[41]. Imagenet dataset is of size 140 GB with 1.2 million images with 1000 classes. The UCF101 is an action recognition data set of realistic action videos, collected from YouTube, having 101 action categories and 13320 videos.

5.2 Maximum Batch Size

In this section, we will try to estimate maximum batch sizes for different models and run different experiments to check till what batch size the model actually ran and how close was the estimate.

For each of the model, the model was run for batch size 10 and batch 50 for 20 iterations. The memory usage by cuda was measured by putting the `torch.cuda.memory_cached` counter in the training iteration loop. The

memory usage was found to be constant over the later 15 iterations and the average of it was used in calculations. The input and output size of the variables was also computed using the methods mentioned in section 4.1. Once these numbers were measured, the linear equations were solved as mentioned in the section 4.1 and maximum batch size, $(B_m)_{estimated}$ was computed.

To get the actual maximum batch size, the model was run for increasing batch sizes until the run failed with a Runtime error of memory out of bounds for GPU. The batch size till it ran without errors was computed as to be actual maximum batch size, $(B_m)_{actual}$.

The percentage error between the estimated and actual maximum batch size is computed using:

$$Error(\%) = \frac{(B_m)_{actual} - (B_m)_{estimated}}{(B_m)_{actual}} \tag{5.1}$$

As the maximum batch size number correctness is not related to the performance of the model for this evaluation, the model was run only for 1 epoch to examine whether the current batch size is feasible or not. All models were run on 1 GPU as we are trying to compute the maximum batch size per GPU. The models were run with 0 number of workers. The number of workers does not play a role in the maximum batch size correctness evaluation. The results presented in table 5.1 are from runs performed on machine M1.

Model	$(B_m)_{estimated}$	$(B_m)_{actual}$	**Error(%)**
Resnet34	373	450	17.1
Resnet50	158	176	10.22
Resnet152	76	80	5
Densenet201	71	71	0
Alexnet	2115	2875	26.4
Conv3d	191	210	9.04

Table 5.1: **Estimated and actual maximum batch size for different CNN models.** The table shows the actual maximum batch size possible, estimated number for maximum batch size and the percentage of error between them for 6 different CNN models.

As we can see from Table 5.1 there is a maximum error of 26.4 % in the estimate. This can be explained by the overestimation we used in the algorithm. As we did use the upper bounds for parameters and feature maps memory usage, it gave us a ballpark estimate of the maximum batch size. Our estimate of maximum batch size is always within the limits of the actual maximum batch size. Also, we can see that as equivalent are the sizes of layers in the model, the closer will be our estimate for example in the case of Densenet201 and Resnet152.

We also experimented with different GPU memory system. The same runs were done on both machine M1 and M2.

The table 5.2 shows how the estimated batch size changes with the

Model	$(\mathrm{B}_m)_{estimated}\|11GB$	$(\mathrm{B}_m)_{estimated}\|16GB$
Resnet34	230	373
Resnet50	102	158
Resnet152	50	76
Densenet201	46	71
Alexnet	1384	2115

Table 5.2: **Estimated maximum batch size for 2 different GPU memory sizes.** The table illustrates the estimation of maximum batch size dependence on GPU memory. The maximum batch size is smaller for smaller GPU memory.

requirements. In the evaluation, the available GPU memory was the same as the total GPU memory.

5.3 Workers and Batch size estimate

In this section we are trying to estimate the minimum number of workers needed while maximizing the batch size under the constraint that batch size is within the maximum batch size, GPUs are not bottlenecked for data and workers are such that there is no memory bound error for DRAM. The number of workers is minimum to attain optimal time but at the same time not wasting CPU cycles if there is no significant benefit of having more workers.

To estimate the number of workers and batch size, we first computed the maximum batch size for a given configuration of machine and model. The iterations the model ran for to get the memory performance data for to com-

pute the maximum batch size, was also used to gather the information of time taken by GPU to run a batch i.e. t_g and the time taken to get the batch size amount of data that is t_d. These numbers were used to solve the Lagrange multiplier problem as mentioned in section 4.2.

To evaluate how good the estimate was we ran the models for 3 epochs on the estimated workers and batch size. Training time per epoch and average GPU Utilization were collected for the runs. Similarly, the models were also run for the cases when the workers are the same but the batch size slightly higher and lower than the current batch size. The models were also run for the cases when the batch size was the same as estimated and the number of workers was varied on the lower and higher end. These numbers were gathered and are reported for three different models namely Resnet34, Resnet50, and Conv3d.

In case of Resnet50, from the table 5.3 we can see that the estimated numbers(in bold) is close to optimal. We got as minimal time as other combination with a minimal number of workers and at the same time keeping the GPU utilization to the maximum of 98%. As the batch size was equal to the maximum batch size estimated, increasing the batch wasn't an option.

For Conv3d resnet model, as shown in the table 5.4 we can infer that estimated numbers(in bold) performed comparable and are approximate to

Workers	**Batch Size**	**Training Time per epoch(seconds)**	**Avg GPU Utilization(%)**
64	160	3544.27	98.54
32	**160**	**3546.12**	**98.71**
16	160	3558.64	98.74
32	128	3549.33	98.52
64	128	3528.58	98.61

Table 5.3: **Resnet50 - training time per epoch and avg. GPU utilization for set of workers and batch sizes.** The table shows the training time epoch and average GPU utilization in case of estimated numbers(bold) from the algorithm performed optimally as compared to other number of workers and batch sizes for model Resnet50.

the optimal numbers. Reducing the number of workers decreased GPU Utilization whereas more workers were not feasible to run. Reducing the batch on the same workers increased the time and on higher workers was also not significantly beneficial.

In case of Resnet34, from the table 5.5 it is evident that the numbers estimated(in bold) are a close approximation. However, 16 workers were optimal in this case. Increasing the workers increased the time. Decreasing the batch size did reduce the time but so the utilization. The estimated numbers worked decently in terms of training time and GPU Utilization.

The table 5.6 shows that as the memory was reduced, the number of workers reduced and so did the batch size. This supports the fact that Work-

Workers	Batch Size	Training Time per epoch(seconds)	Avg GPU Utilization(%)
32	190	N/A	N/A
16	**190**	**683**	**27.5**
8	190	635	24.6
16	128	778	26.7
32	128	640	24.49

Table 5.4: **Conv3d - training time per epoch and avg. GPU utilization for set of workers and batch sizes.** The table shows the training time epoch and average GPU utilization in case of estimated numbers(bold) from the algorithm performed close to optimal as compared to other number of workers and batch sizes for model Conv3d resnet.

ers are limited because of DRAM size, the lower the DRAM the fewer workers could be accommodated for reasonable batch size.

5.4 CNN model performance measure

In this section, we tried to evaluate our number in terms of Machine Learning performance measures. The models were run on default batch size as mentioned in their GitHub code or their paper and the batch size that was estimated from the algorithm. The workers were kept the same for both default and estimated numbers to maintain generality. The models were run for 50 and 35 epochs respectively rather till convergence to measure the goodness of the model at a certain point. The validation accuracy and validation loss were measured at the end of the epochs and also the training time was noted for the same. The models for both the default and estimated batch sizes were

Workers	Batch Size	Training Time per epoch(seconds)	Avg GPU Utilization(%)
64	373	1738.24	96.85
32	**373**	**1714.83**	**98.07**
16	373	1710.07	98.37
32	256	1681.3	97.86
64	256	1714.33	97.09

Table 5.5: **Resnet34 - training time per epoch and avg. GPU utilization for set of workers and batch sizes.** The table shows the training time epoch and average GPU utilization in case of estimated numbers(bold) from the algorithm performed a close to optimal as compared to other number of workers and batch sizes for model Resnet34.

Model	Memory(GB)	Workers	Batch Size
Resnet50	32	16	101
Resnet50	128	32	160
Resnet152	32	32	50
Resnet152	128	64	78

Table 5.6: **Variation of estimated numbers of workers and batch size on DRAM size.** The table shows the change in the estimated number of workers and batch size as the DRAM size is different. The workers are more for larger DRAM size of 128 GB.

run on the same machine.

As we can see from the table 5.7, in case of Conv3d resnet model which was used for video activity recognition, after 50 epochs not only at the estimated batch size the training time was less but also the accuracy and loss weren't compromised. Both the batch size had equivalent average validation loss and accuracy. Though the estimated batch size was 190, the program was

Model	Batch Size	Validation Loss	Validation Accuracy	Time (minutes)
Conv3d	155	3.212	30.8	190
	16	3.273	31.8	220
Resnet50	373	7.18	0.0	1018.9
	128	7.35	0.0	1034.0

Table 5.7: **Comparison of validation accuracy and validation loss for estimated numbers by algorithm with default numbers of workers and batch size.** The table shows that estimated numbers performed well with respect to validation accuracy and loss as compared to default numbers for two different models when ran for same number of epochs.

run for 155 batch sizes as the validation set required a large extra amount of space as were videos.

In the case of Resnet50, the model with batch size 373, estimated from algorithm had lesser validation loss and it achieved the same in a smaller amount of time as compared to the default batch size. One thing to notice is as epochs will increase the time difference between the both is going to increase as per iteration time for batch size 128 is more by around ~ 34 seconds as compared to batch size 373.

5.5 Manual Search vs Algorithm

In this section, we tried to compare the benefit of using the algorithm with respect to the general case of manual search for batch size.

The first thing we want to point is choosing numbers for batch size, ML engineers often search for batch size as a power of 2 whereas there has been no evidence that batch size needs to be a power of 2. We experimented with multiple unorthodox batch size values and they performed well with respect to training time per epoch, utilization and also validation accuracy and loss as reported in above section 5.4. Our estimation of numbers explores those values as well which are not part of a manual search.

5.5.1 Manual Search

In the case of manual search, the general way is to start with numbers as low as 32 and increase it by a factor of two until it runs out of memory. To check if a particular batch size ran, the model should are run for at least 0.5% to 2% of the total number of iterations to confirm if it actually works. The number of iterations could be less to check if the memory consumption is considerably low for example 10% to 30% of the available GPU memory.

So for example, say for a model M, a manual search might work as, running for some i iterations for batch size $32(2^5)$ then for some j iterations for batch size $64(2^6)$ and so on till it runs out of memory for batch size $B(2^N)$.

Total time to find a batch could be formulated as:

$$T = \sum_{i=5}^{N} t_i * n_i \tag{5.2}$$

where t_i is the time taken per iteration for batch size 2^i and n_i is the number of iterations the model ran to check the feasibility of batch size 2^i.

5.5.2 Algorithm

In the case of the algorithm, the model needs to be run for 20 iterations for two different batch sizes as small as 10 and 50, which doesn't take a significant amount of time. Once it is run for 20 iterations, the algorithm will tell the maximum batch size possible. For better performance with respect to resources, it would tell an approximate amount of workers and batch size as well. These numbers could be used directly for training purpose without further tweaking.

5.5.3 Comparison

For the comparison between manual search and algorithm, we have taken into account the process described in section 5.5.1 as manual search, it is one way to do the manual search and is subjective with respect to the person thus the numbers might not always be the same. We also assumed that the numbers computed by the algorithm are used directly without further tweaking thus the time to find batch size in case of the algorithm is just

the time to run 20 iterations for the algorithm at two different batch sizes and the algorithm to run on the numbers. In the cases of manual search, it is run with 16 number of workers and the same number of GPUs that is 1. In the case of the algorithm, as to estimate the number of workers, we need time to get data without prefetching thus the 20 iterations are run with 0 number of workers.

Model	**Time**$_{\text{manual}}$ **(seconds)**	**batch size**$_{manual}$	**Time**$_{\text{algo}}$ **(seconds)**	**batch size**$_{algo}$
Resnet34	887.85	256	41.14	230
Densenet201	94	32	134.35	46
Alexnet	5133.51	1024	40.25	1384

Table 5.8: **Comparison of time taken to search for batch size in case of Manual search vs Algorithm.** The table shows that the algorithm is faster or comparable to reach to batch size estimate as compared to manual search for different models.

As shown in table 5.8, we can see that the algorithm is comparatively faster in the case when the batch size lies in the higher range, whereas it is comparable in case of smaller batch size range but still comparable.

In the case of Resnet34, as the estimation has 10% error our estimation is lower than the batch size it could run for. Though the time to find that batch size manually is 20 times more than the algorithm. For Densenet on the other hand, as we need to run for 10 and 50 batch size initially and they run

on 0 number of workers, algorithm took slightly more time than the manual but it was able to get the larger batch size of 46. For a network like Alexnet where higher batch sizes are possible, it took much more time manually to get the larger batch size possible than the algorithm.

Consequently, the time noted here are dependent on the available resources. For example, if the GPU memory is less, the time taken per iteration is going to be more which will affect the manual search time by at least 10 times more than the algorithm as it needs to run for around 100 iterations to infer if that batch size is feasible. Also, in the case when models are large it might take more time to run iterations which will again affect the manual search time significantly more.

Chapter 6

Limitations

This chapter proposes some of the limitations of the proposed work.

We have tried to analyze the convolutional neural networks as they are the ones which have high data input and storage systems play a significant role in the training.

The methodologies we have come up with for the estimation of the maximum batch size and the workers are limited for the convolutional neural network. The method used might not be generalized for other neural networks like RNN(Recurrent Neural Network) as they use separate memory units called LSTMs which are not used in CNNs, that makes RNNs different from CNNs and thus will have different GPU memory usage.

Also, in the problem of estimating the number of workers and the batch size, we try to maximize the batch size though it is possible that larger batch size may not always help in convergence. The larger the batch size, the lower will be the iterations per epoch and thus the number of gradient updates. For a model to train properly, there has to be a certain number of updates

which the model should have per epoch. This number is not a deterministic number and thus hard to estimate the largest batch size one can use. But from the experiments and evaluation, we might say that it depends on the dataset size and if the dataset is big enough that even for larger batch sizes, it gets the sufficient amount of updates then one could use the larger batch size.

We did our experiments on widely used PyTorch platform. Though analysis which led to the development of algorithm was dependent on PyTorch, the methodology proposed is not dependent on the platform. It is dependent on the available resources and model, thus we expect it to generalize over other platforms.

Chapter 7

Conclusion

In the thesis, we analyzed the CNN model, how they are trained and how the different system aspects affect the performance of CNN model in terms of training time and resource utilization. We found out that for better utilization of resources, one has to find an optimal combination of batch size and the number of workers.

Finding the batch size and number of workers comes under the umbrella of hyperparameter optimization. We took one step forward in reducing the search space for a feasible set of batch size. The work is limited to CNNs and estimates the maximum batch size given the model data and resources available. We tried to estimate a combination of workers and batch size for different CNN models.

The evaluation shows that the estimate of maximum batch size shows that even for higher batch size the error is less than 30% and for lower ranges, it is as close to the actual maximum batch size. The estimated number of workers and batch size were close to good performance as the training time

was comparative and GPU utilization was more than 80%.

We also showed that some of the CNN models performed better with larger batch sizes as compared to their default batch size. They were able to achieve comparative validation loss and accuracy in a lesser amount of time.

Bibliography

[1] I. Kononenko, "Machine learning for medical diagnosis: history, state of the art and perspective," *Artificial Intelligence in medicine*, vol. 23, no. 1, pp. 89–109, 2001.

[2] G. D. Magoulas and A. Prentza, "Machine learning in medical applications," in *Advanced Course on Artificial Intelligence*. Springer, 1999, pp. 300–307.

[3] P. Maes and R. A. Brooks, "Learning to coordinate behaviors." in *AAAI*, vol. 90, 1990, pp. 796–802.

[4] N. Kohl and P. Stone, "Machine learning for fast quadrupedal locomotion," in *AAAI*, vol. 4, 2004, pp. 611–616.

[5] M. L. Littman, "Value-function reinforcement learning in markov games," *Cognitive Systems Research*, vol. 2, no. 1, pp. 55–66, 2001.

[6] L. R. Rabiner and B.-H. Juang, "An introduction to hidden markov models," *ieee assp magazine*, vol. 3, no. 1, pp. 4–16, 1986.

[7] X. Ma and E. Hovy, "End-to-end sequence labeling via bi-directional lstm-cnns-crf," *arXiv preprint arXiv:1603.01354*, 2016.

[8] K. Tomanek and U. Hahn, "Semi-supervised active learning for sequence labeling," in *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*. Association for Computational Linguistics, 2009, pp. 1039–1047.

[9] R. S. Sutton, A. G. Barto *et al.*, *Introduction to reinforcement learning*. MIT press Cambridge, 1998, vol. 135.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[11] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[14] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[15] Hyperparameter (machine learning), "Hyperparameter (machine learning) — Wikipedia, the free encyclopedia," [Online; accessed May 6, 2019]. [Online]. Available: https://en.wikipedia.org/wiki/Hyperparameter_(machine_learning)

[16] A. Gozzoli. (2018) Practical guide to hyperparameters search for deep learning models. [Online]. Available: https://blog.floydhub.com/guide-to-hyperparameters-search-for-deep-learning-models/

[17] AlexNet, "Alexnet — Wikipedia, the free encyclopedia," [Online; accessed May 6, 2019]. [Online]. Available: https://en.wikipedia.org/wiki/AlexNet

[18] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017.

[19] P. M. Radiuk, "Impact of training set batch size on the performance of convolutional neural networks for diverse datasets," *Information Technology and Management Science*, vol. 20, no. 1, pp. 20–24, 2017.

[20] Artificial neural network, "Artificial neural network — Wikipedia, the free encyclopedia," [Online; accessed May 6, 2019]. [Online]. Available: https://en.wikipedia.org/wiki/Artificial_neural_network

[21] Perceptron, "Perceptron — Wikipedia, the free encyclopedia," [Online; accessed May 6, 2019]. [Online]. Available: https://en.wikipedia.org/

wiki/Perceptron

[22] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural networks for perception.* Elsevier, 1992, pp. 65–93.

[23] Deep Learning, "Deep learning — Wikipedia, the free encyclopedia," [Online; accessed May 6, 2019]. [Online]. Available: https://en.wikipedia.org/wiki/Deep_learning

[24] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249–256.

[25] S. Saha. (2018) A comprehensive guide to convolutional neural networks—the eli5 way. [Online]. Available: https://blog.floydhub.com/guide-to-hyperparameters-search-for-deep-learning-models/

[26] Recurrent neural network, "Recurrent neural network — Wikipedia, the free encyclopedia," [Online; accessed May 6, 2019]. [Online]. Available: https://en.wikipedia.org/wiki/Recurrent_neural_network

[27] Hyperparameter optimization, "Hyperparameter optimization — Wikipedia, the free encyclopedia," [Online; accessed May 6, 2019]. [Online]. Available: https://en.wikipedia.org/wiki/Hyperparameter_optimization

[28] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012.

[29] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter, "Fast bayesian optimization of machine learning hyperparameters on large datasets," *arXiv preprint arXiv:1605.07079*, 2016.

[30] Tensorflow. Tensorflow. [Online]. Available: https://www.tensorflow.org/

[31] PyTorch. Pytorch. [Online]. Available: https://pytorch.org/

[32] Caffe. Caffe. [Online]. Available: https://caffe.berkeleyvision.org/

[33] Keras. Keras. [Online]. Available: https://keras.io/

[34] MxNet. Mxnet. [Online]. Available: https://mxnet.apache.org/

[35] PyTorch. Source code for torch.utils.data.dataloader. [Online]. Available: https://pytorch.org/docs/stable/_modules/torch/utils/data/dataloader.html

[36] ——. Multiprocessing package - torch.multiprocessing. [Online]. Available: https://pytorch.org/docs/stable/multiprocessing.html

[37] ——, "Imagenet training in pytorch," https://github.com/pytorch/examples/tree/master/imagenet, 2013.

[38] P. U. Stanford Vision Lab, Stanford University. (2016) Imagenet dataset. [Online]. Available: http://image-net.org/about-overview

[39] PyTorch. Cuda semantics. [Online]. Available: https://pytorch.org/docs/stable/notes/cuda.html#memory-management

[40] Lagrange multiplier, "Lagrange multiplier — Wikipedia, the free encyclopedia," [Online; accessed May 6, 2019]. [Online]. Available: https://en.wikipedia.org/wiki/Lagrange_multiplier

[41] K. Soomro, A. R. Zamir, and M. Shah, "Ucf101: A dataset of 101 human actions classes from videos in the wild," *arXiv preprint arXiv:1212.0402*, 2012.

[42] J. M. Zurada, *Introduction to artificial neural systems*. West publishing company St. Paul, 1992, vol. 8.

[43] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in neural information processing systems*, 1990, pp. 396–404.

[44] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*, 2014.

[45] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings*

of the 25th international conference on Machine learning. ACM, 2008, pp. 160–167.

[46] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[47] D. P. Mandic and J. Chambers, *Recurrent neural networks for prediction: learning algorithms, architectures and stability*. John Wiley & Sons, Inc., 2001.

[48] Long Short term memory, "Long short-term memory — Wikipedia, the free encyclopedia," [Online; accessed May 6, 2019]. [Online]. Available: https://en.wikipedia.org/wiki/Long_short-term_memory

[49] D. A. Hernandez. (2018) Ai and compute. [Online]. Available: https://openai.com/blog/ai-and-compute/

[50] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems*, 2012, pp. 2951–2959.

[51] PyTorch. Data loading and processing tutorial. [Online]. Available: https://pytorch.org/tutorials/beginner/data_loading_tutorial.html

[52] M. Zeunert. A super simple introduction to neural networks. [Online]. Available: https://www.mattzeunert.com/2016/12/09/neural-networks-super-simple-introduction.html

[53] M. Agarwal. Back propagation in convolutional neural networks—intuition and code. [Online]. Available: https://becominghuman.ai/back-propagation-in-convolutional-neural-networks-intuition-and-code-714ef1c38199

[54] B. Boehmke. Feedforward deep learning models. [Online]. Available: http://uc-r.github.io/feedforward_DNN

[55] S. Saha. A comprehensive guide to convolutional neural networks—the eli5 way. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[56] D. De. Rnn or recurrent neural network for noobs. [Online]. Available: https://hackernoon.com/rnn-or-recurrent-neural-network-for-noobs-a9afbb00e860

[57] J. Jordan. Common architectures in convolutional neural networks. [Online]. Available: https://www.jeremyjordan.me/convnet-architectures/

Vita

Aastha Tripathi was born in Indore in the state of Madhya Pradesh in India, the daughter of Mr. Laxmikant Tripathi and Mrs. Manjula Tripathi. She did her schooling in Shri Gujarati Samaj School in Indore. She secured rank 570 in all India engineering exam called IIT-JEE among 0.5 million students.

She received her Bachelor of Technology degree in Computer Science and Engineering from Indian Institute of Technology, Delhi. She did a summer internship (Research Intern role) at IBM India Research Labs (in New Delhi) during her Bachelors. After her Bachelors, she worked as Software Development Engineer for one year in Saavn (in Mumbai), which is one of India's biggest music streaming services. She was a part of the Search and Algorithm team in Saavn, which manages working of search and backend development. There she worked in to create and maintain pipeline development of data ingestion process.

She started her Masters in Computer Science at The University of Texas at Austin in Fall 2017. Her master's research was advised by Prof. Vijay Chidambaram and focuses on storage systems, while her areas of interests span

across distributed systems, storage systems, and key-value stores, big-data processing and management, and deep learning. During her Masters, she did her summer internship (Software Engineer role) at Amazon, Austin, as part of the Information Security team. After Masters graduation, she is headed towards Amazon, Palo Alto, where she will be working full-time as Software Engineer.

Permanent address: tripathiaastha68@gmail.com

This thesis was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.